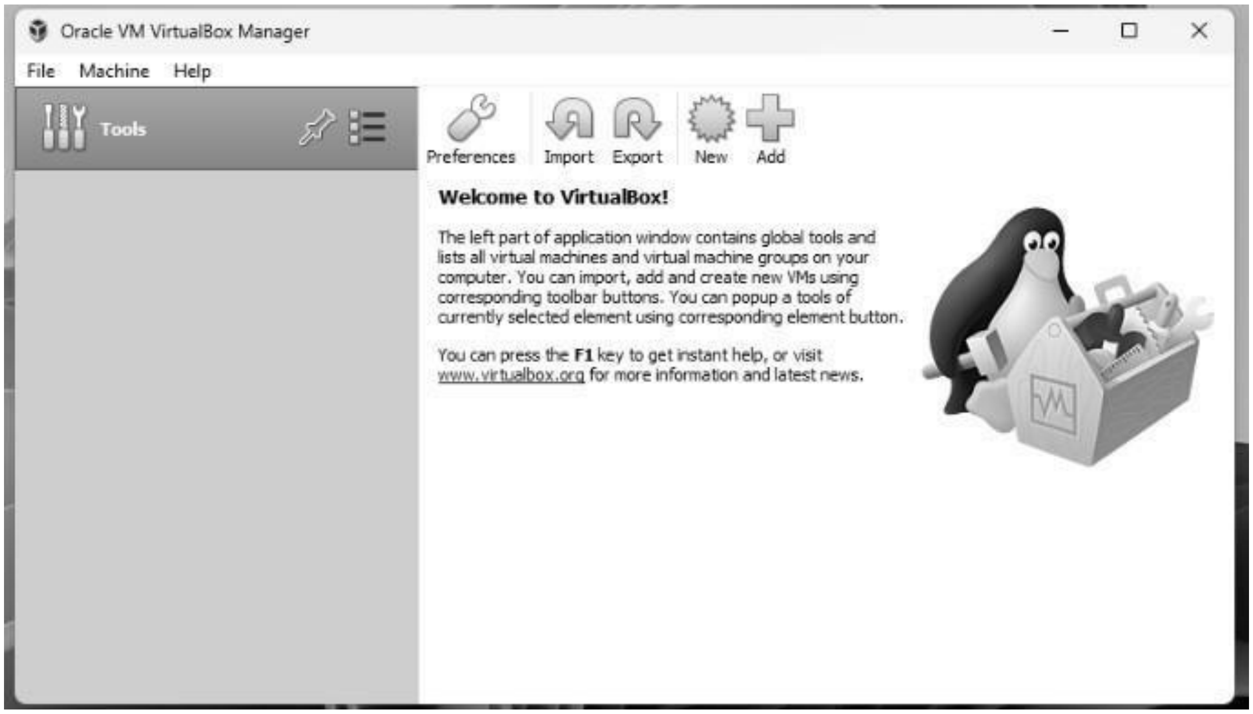
EX NO : 1
DATE:

Installation of Linux/Ubuntu operating system

Step 1: Install VirtualBox from the link below by choosing the suitable host OS:
https://www.virtualbox.org/wiki/Downloads

Step 2: Download the ISO disk image files of the operating systems based on need. In this case,
Ubuntu 16.04 and MS DOS 6.22 from the below links:
https://releases.ubuntu.com/16.04/
https://www.allbootdisks.com/download/iso.html

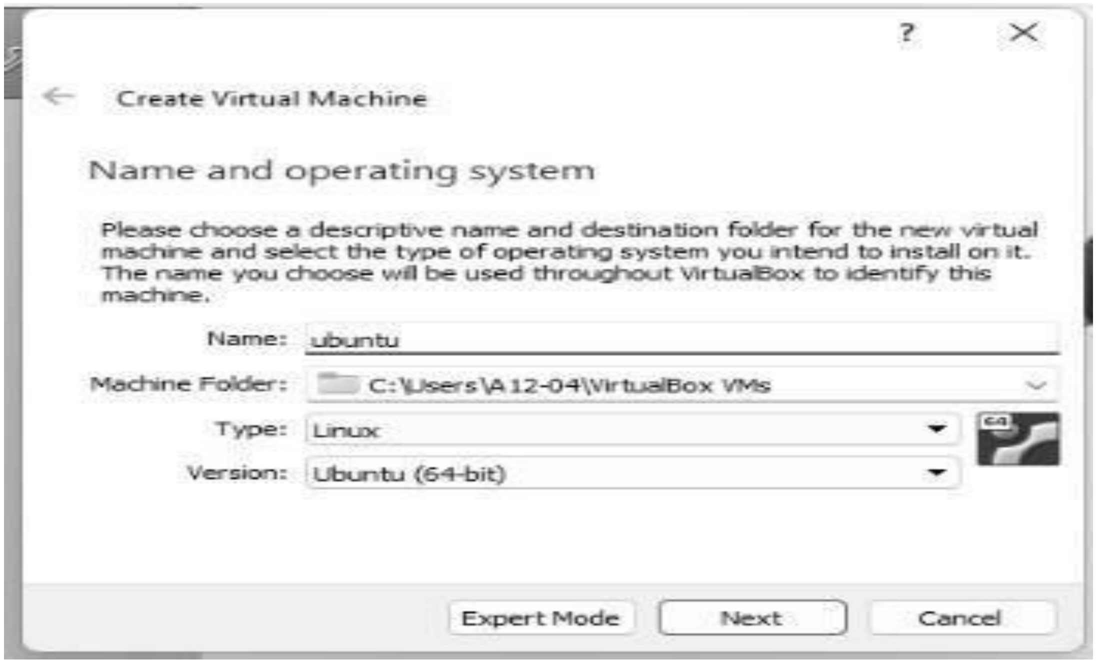


Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

Step 4: Click on New and choose a name and location for the guest OS to be installed. In this case, Ubuntu 16.04 is the name chosen and it is to be stored in the D drive under a folder called VirtualBox. Click Next.



Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

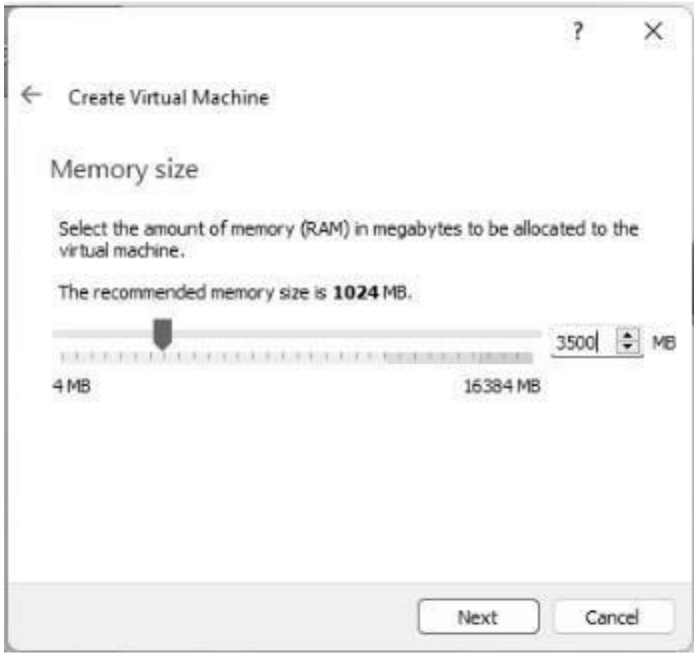
Name: ubuntu

Machine Folder: C:\Users\A12-04\VirtualBox VMs

Type: Linux

Version: Ubuntu (64-bit)

Step 5: Select the amount of memory to be allocated to the virtual machine. In this case, it is 2048 MB (2 GB) Click Next.



Memory size

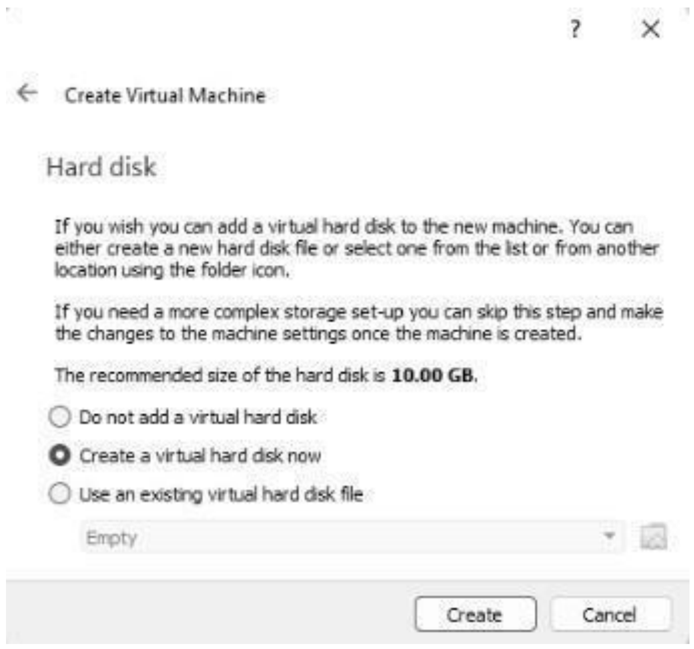
Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

3500 MB

4 MB 16384 MB

Step 6: Create a virtual hard disk for the virtual machine and click Create.



Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

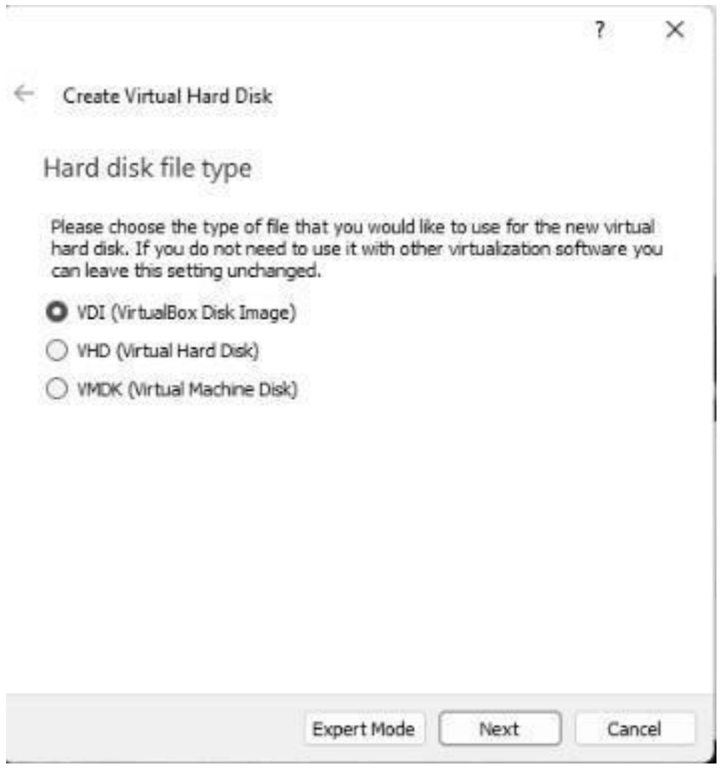
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

- ○ Do not add a virtual hard disk
- ● Create a virtual hard disk now
- ○ Use an existing virtual hard disk file

Empty

Step 7: Select the type of file to be used for the virtual hard disk and click Next. In this case, it is VirtualBox Disk Image (VDI)

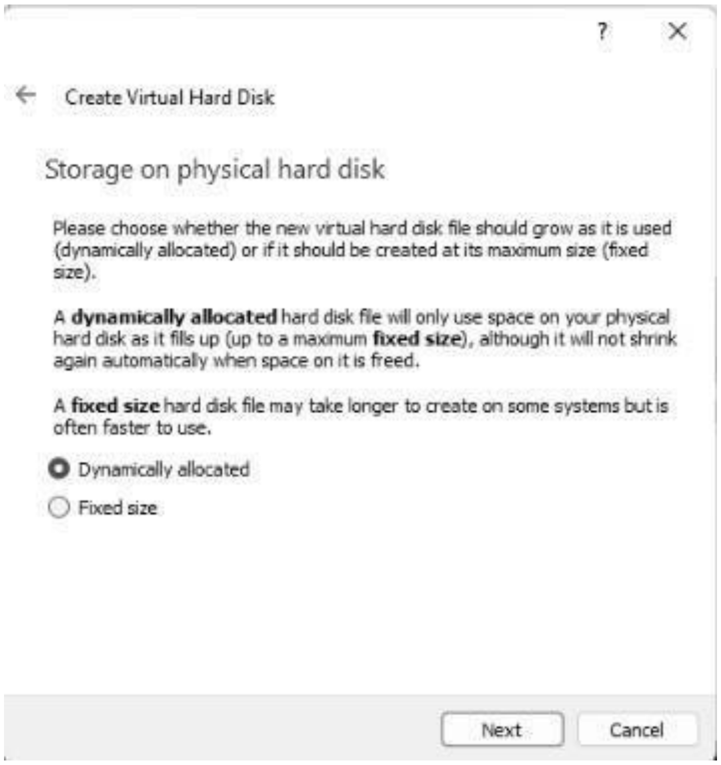


Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- ● VDI (VirtualBox Disk Image)
- ○ VHD (Virtual Hard Disk)
- ○ VMDK (Virtual Machine Disk)

Step 8: Select whether the disk file should be dynamically allocated or be fixed in size. In this case, it is dynamically allocated.



Storage on physical hard disk

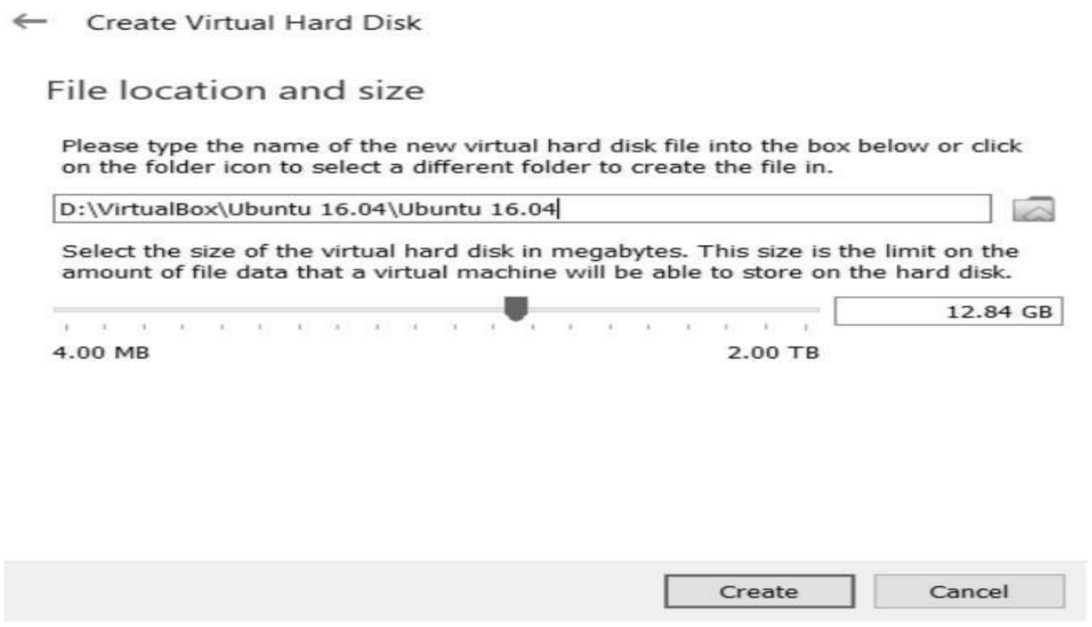
Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

[✓ Dynamically allocated]
Fixed size

Step 9: Select the location and size of the file.



File location and size

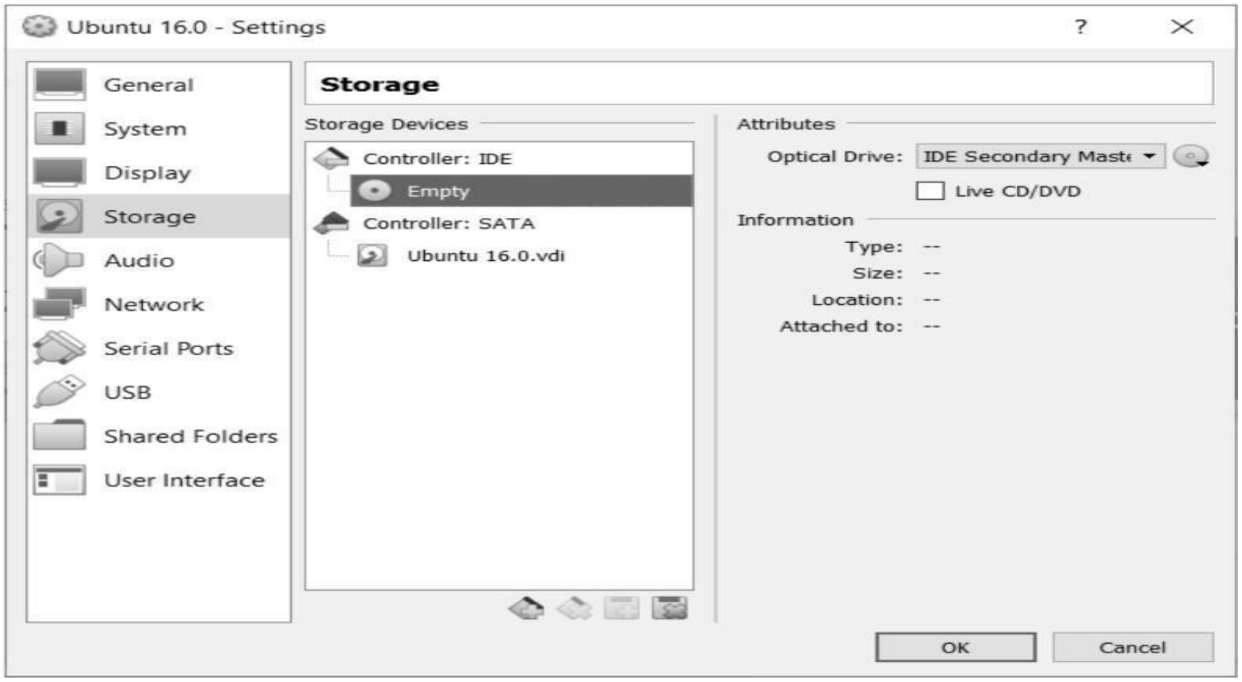
Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

D:\VirtualBox\Ubuntu 16.04\Ubuntu 16.04

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

12.84 GB

Step 10: In Settings, under the storage tab, click on the disc icon and select the ISO Image file that has been downloaded previously. Click on OK.


Ubuntu 16.0 - Settings

Storage

Storage Devices

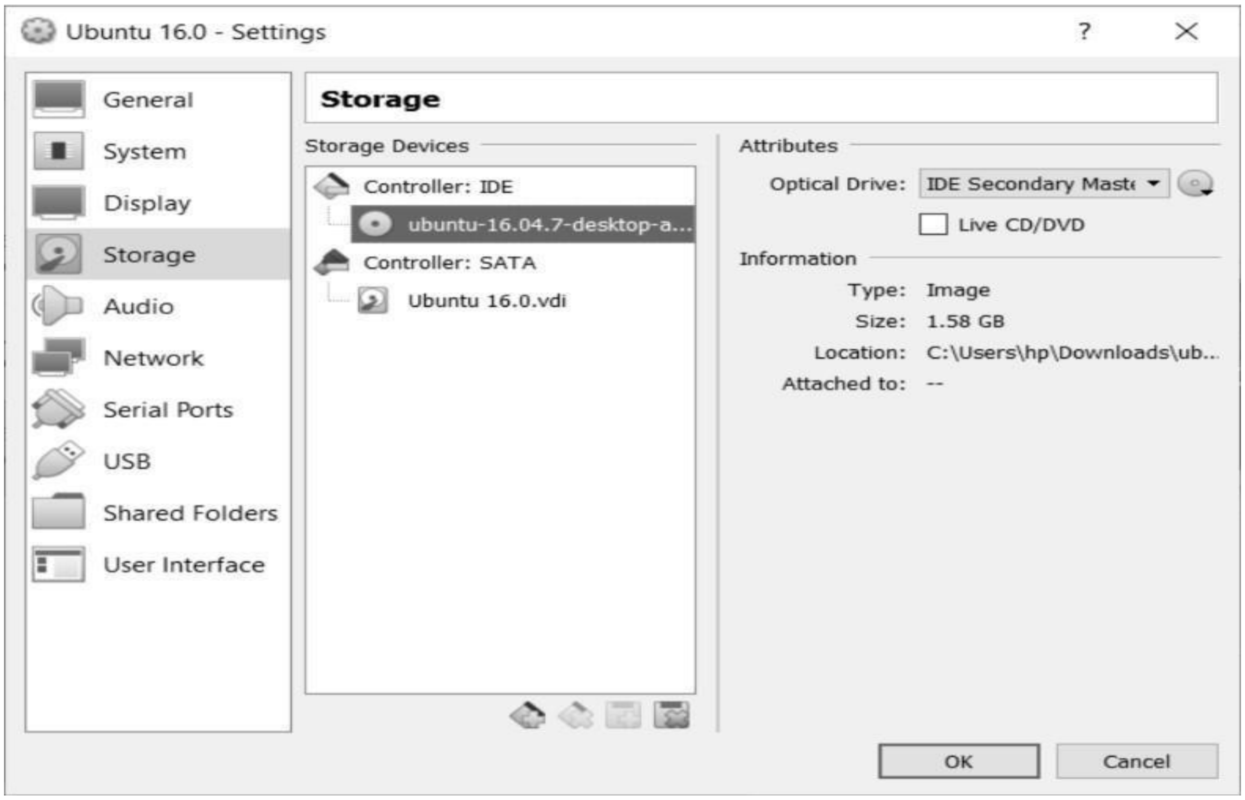
Controller: IDE
Empty
Controller: SATA
Ubuntu 16.0.vdi

Attributes

Optical Drive: IDE Secondary Maste
Live CD/DVD

Information

Type: --
Size: --
Location: --
Attached to: --


Ubuntu 16.0 - Settings

Storage

Storage Devices

Controller: IDE
ubuntu-16.04.7-desktop-a...
Controller: SATA
Ubuntu 16.0.vdi

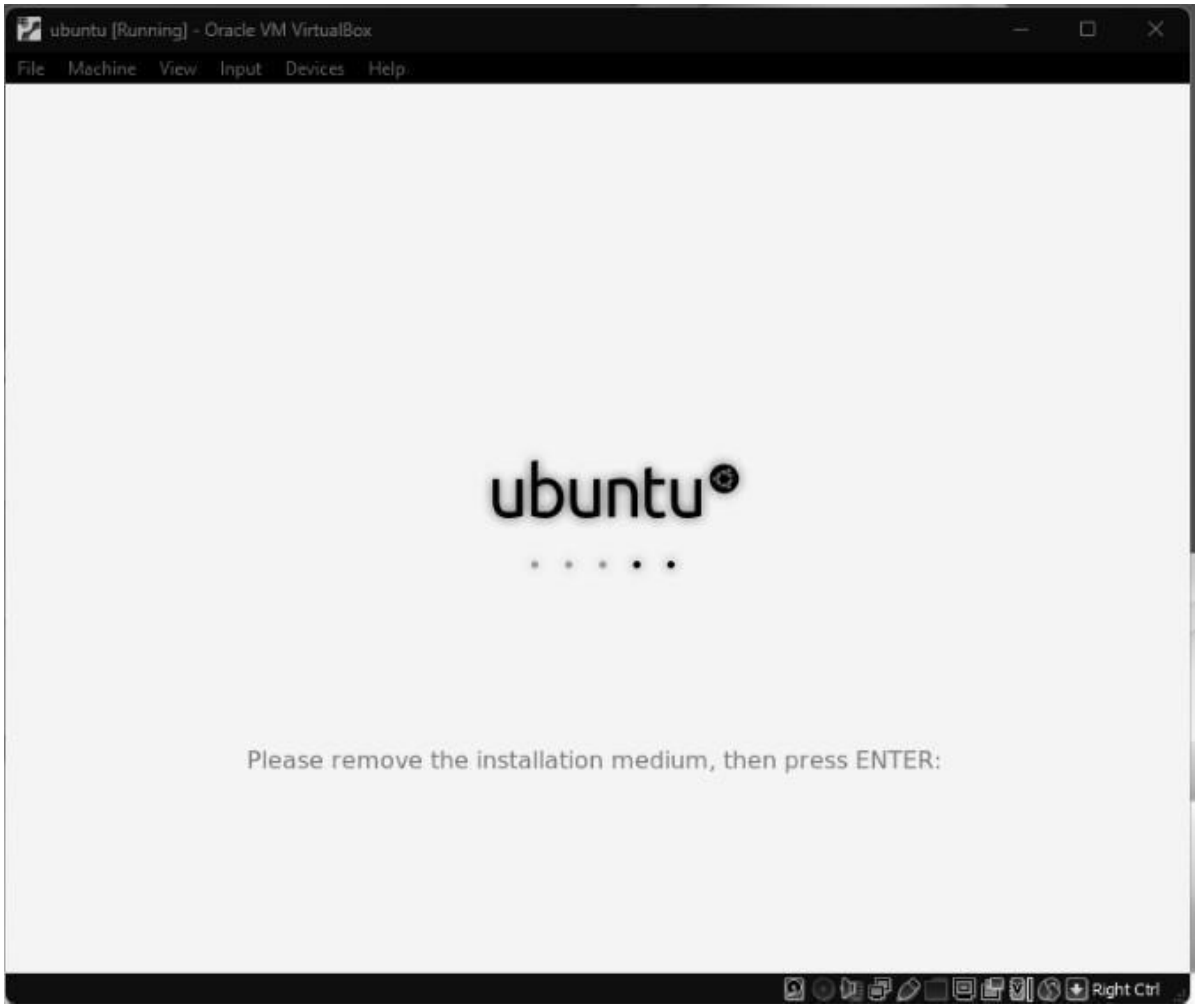
Attributes

Optical Drive: IDE Secondary Maste
Live CD/DVD

Information

Type: Image
Size: 1.58 GB
Location: C:\Users\hp\Downloads\ub...
Attached to: --

Step 11: Start the machine and follow the instructions on the screen to install Ubuntu for the first time. Notice that choosing wipe disk and install option will not wipe out the hard drive of the host system as the virtual machine is unaware of the host system's disk space. In the terminal, run sudo apt install build-essential dkms linux-headers-$(uname -r) command to fill the Ubuntu screen to the entire VirtualBox window.

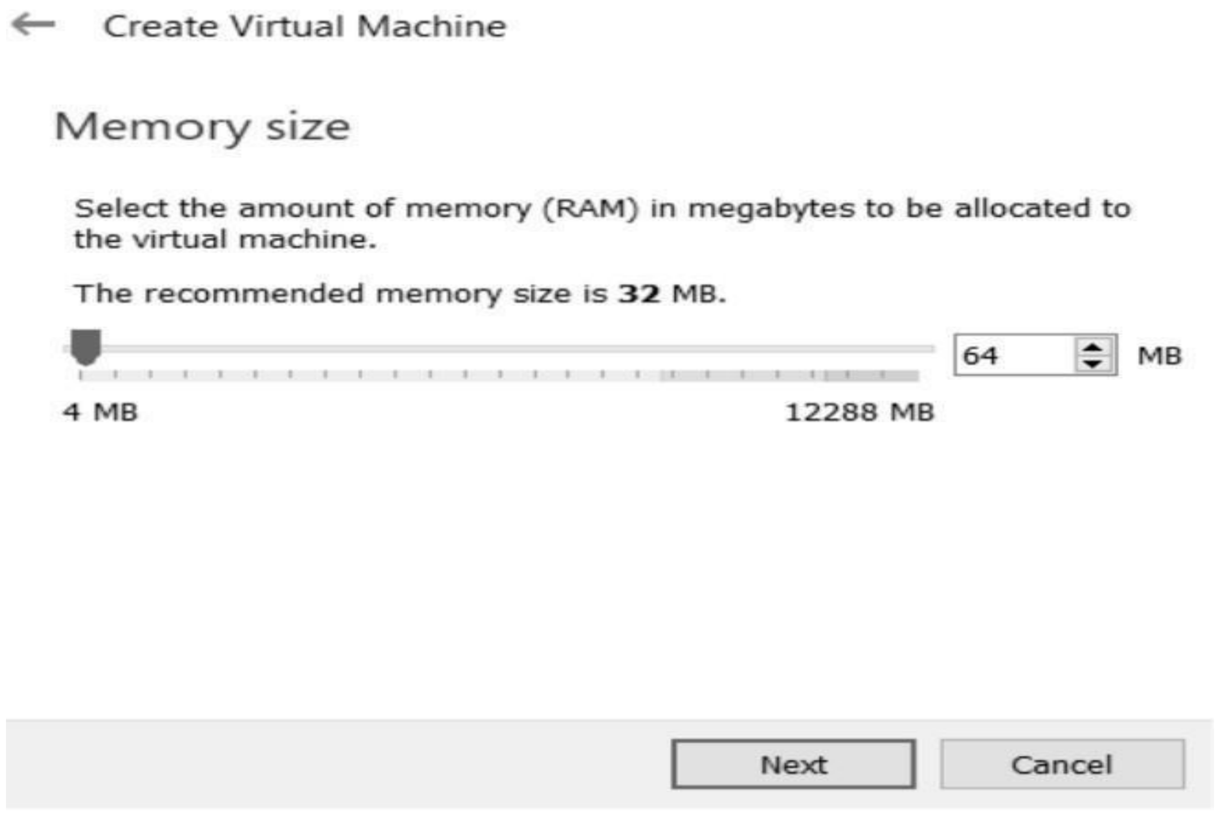


Please remove the installation medium, then press ENTER:

Step 12: Turn off this machine and click on New again and follow the same steps as above to install the MS DOS 6.22 version. The screenshots are attached for each step.

Step 13: Click on New and choose a name and location for the guest OS to be installed.
In this case, MS DOS 6.22 is the name chosen and it is to be stored in the D drive under a folder called VirtualBox. Click Next

Step 14: Select the amount of memory to be allocated to the virtual machine. In this case, it is 64 MB. Click Next.



Create Virtual Machine

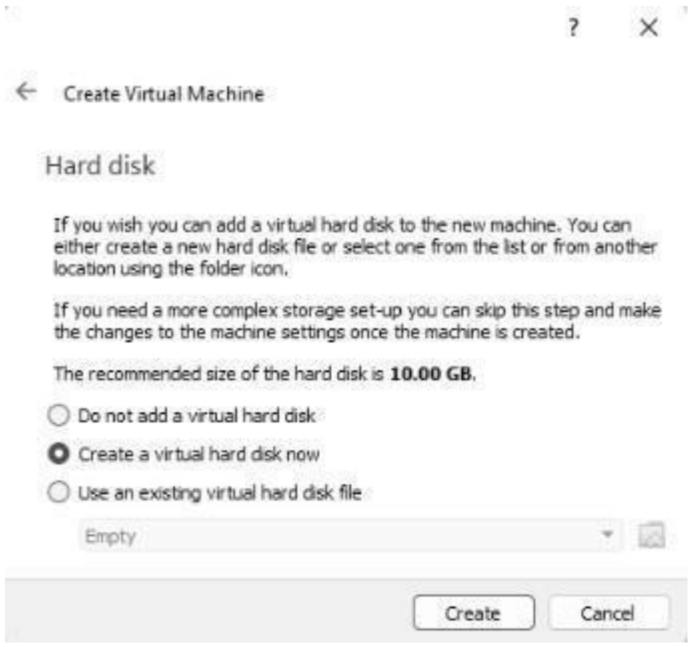
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **32** MB.

64 MB

Step 15: Create a virtual hard disk for the virtual machine and click Create.


Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

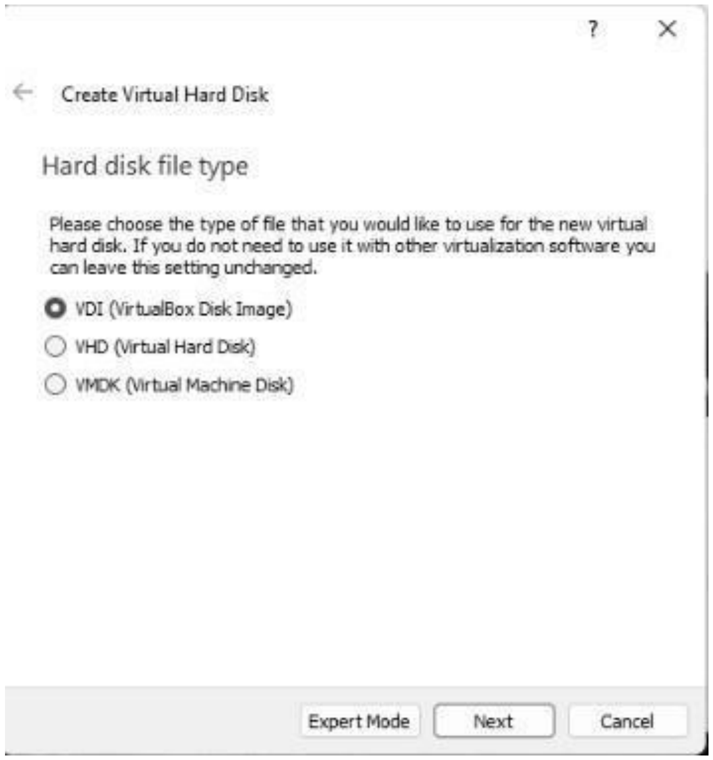
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

- Do not add a virtual hard disk
- [✓ Create a virtual hard disk now]
- Use an existing virtual hard disk file

Empty

Step 16: Select the type of file to be used for the virtual hard disk and click Next. In this case, it is VirtualBox Disk Image (VDI)

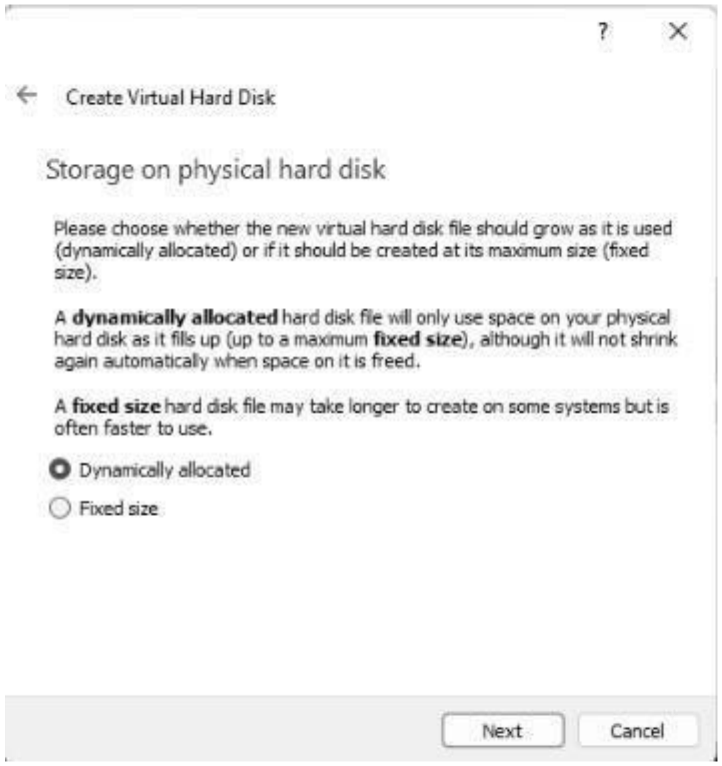

Create Virtual Hard Disk

Hard disk file type

Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

- [✓ VDI (VirtualBox Disk Image)]
- VHD (Virtual Hard Disk)
- VMDK (Virtual Machine Disk)

Step 17: Select whether the disk file should be dynamically allocated or be fixed in size. In this case, it is dynamically allocated.



Storage on physical hard disk

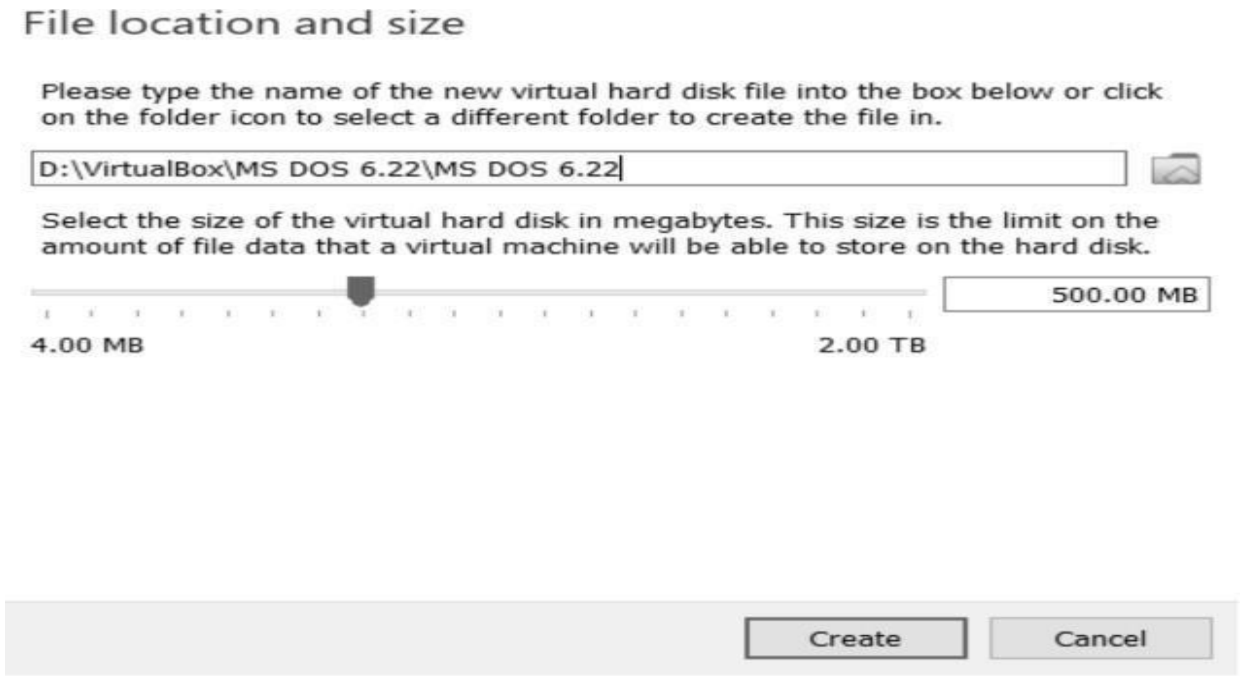
Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

[✓ Dynamically allocated]
Fixed size

Step 18: Select the location and size of the file.



File location and size

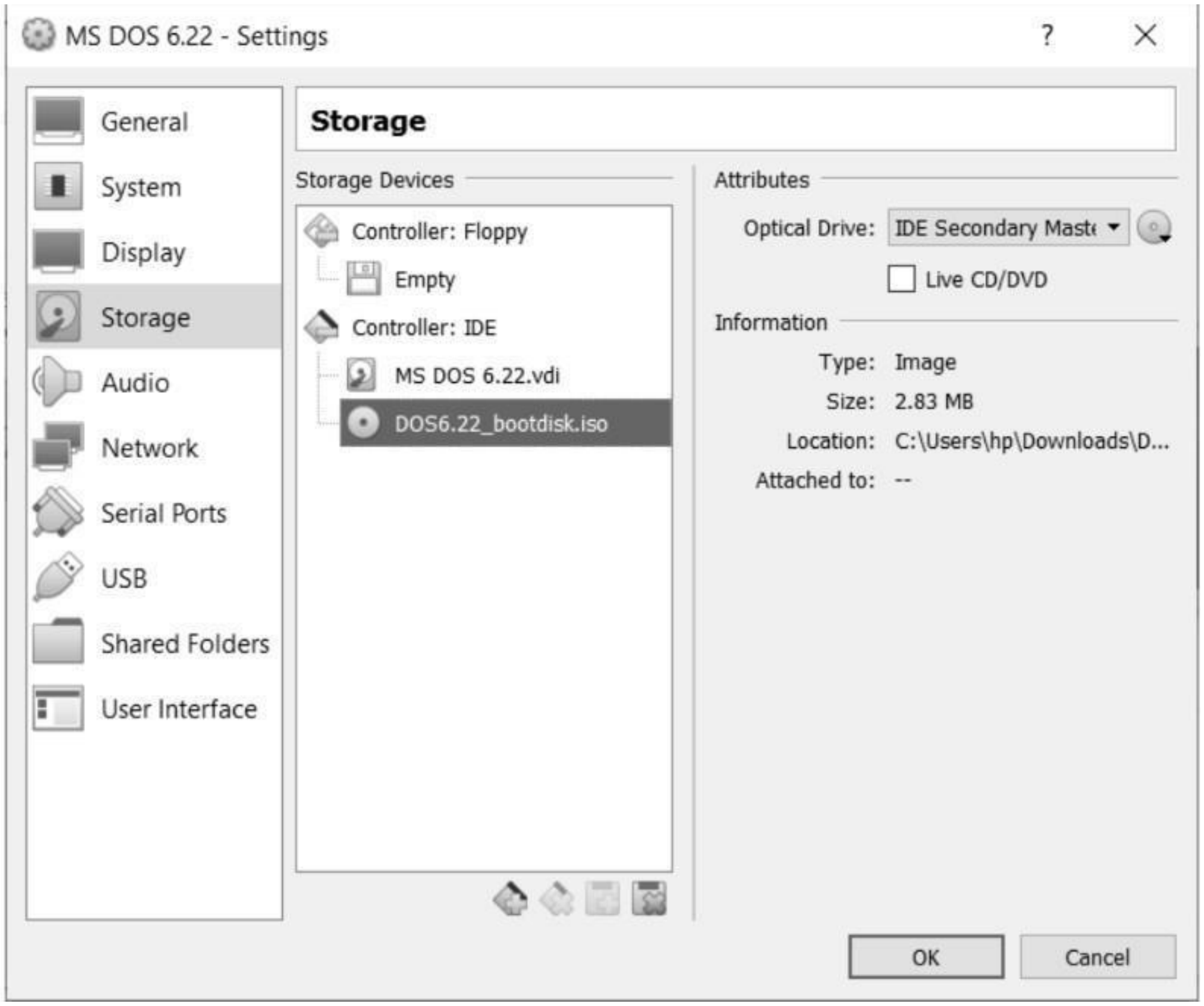
Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

D:\VirtualBox\MS DOS 6.22\MS DOS 6.22

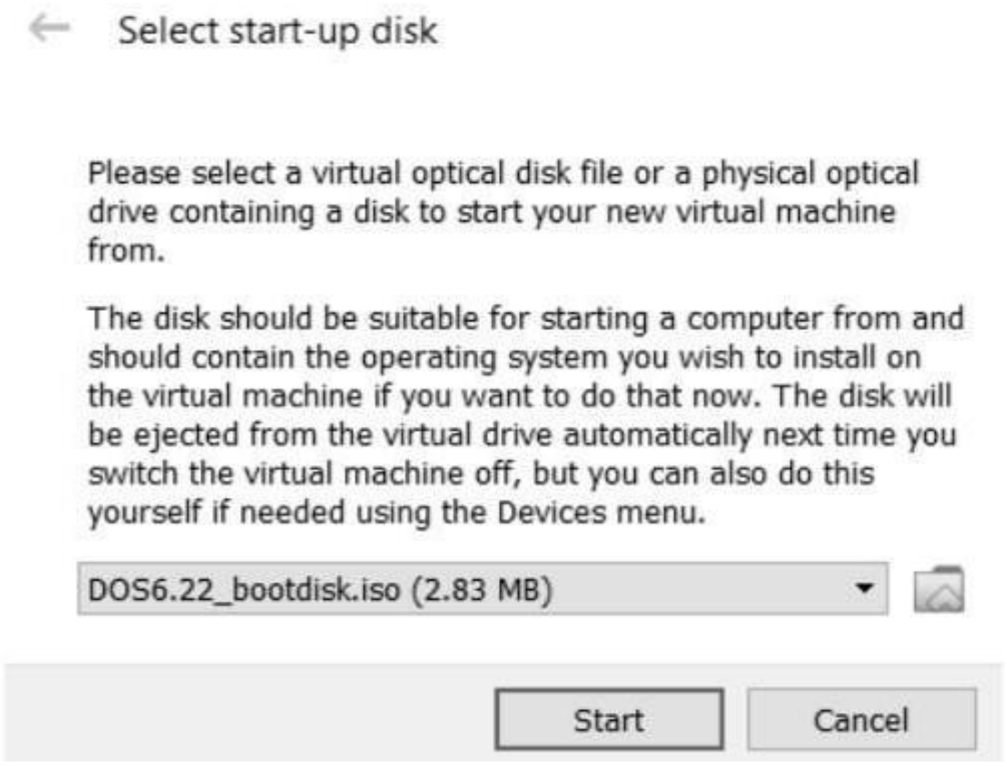
Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

500.00 MB

Step 19: In Settings, under the storage tab, click on the disc icon and select the ISO Image file that has been downloaded previously. Click on OK.



Step 20: Choose the ISO image file location from the dropdown menu in the dialog box that requests for disk startup.

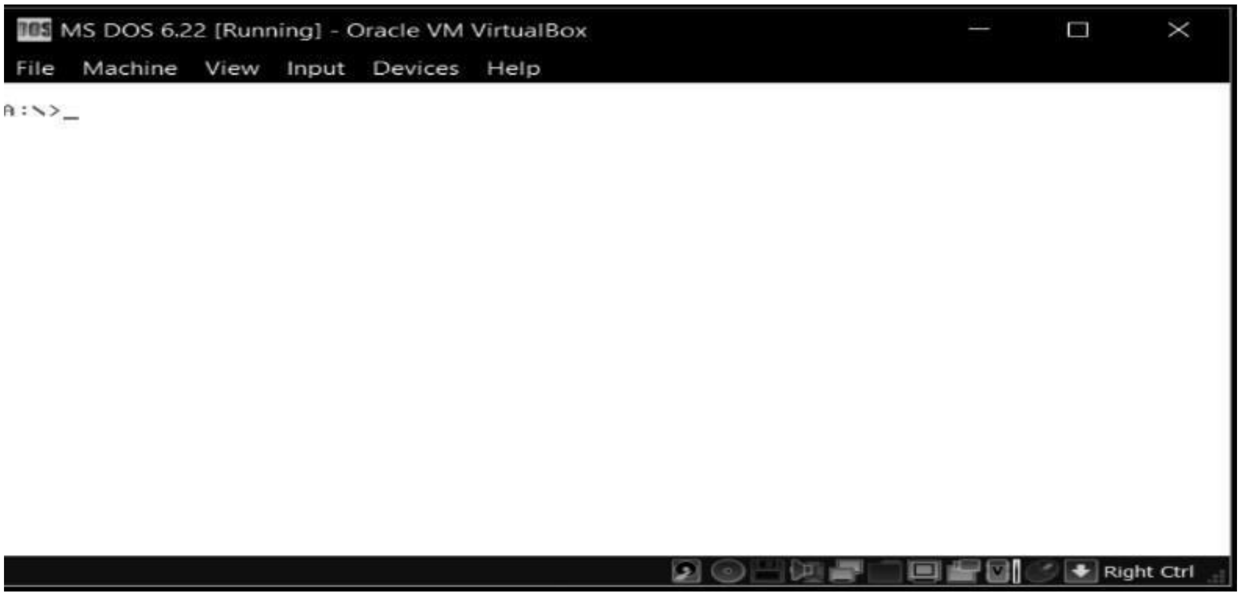

Select start-up disk

Please select a virtual optical disk file or a physical optical drive containing a disk to start your new virtual machine from.

The disk should be suitable for starting a computer from and should contain the operating system you wish to install on the virtual machine if you want to do that now. The disk will be ejected from the virtual drive automatically next time you switch the virtual machine off, but you can also do this yourself if needed using the Devices menu.

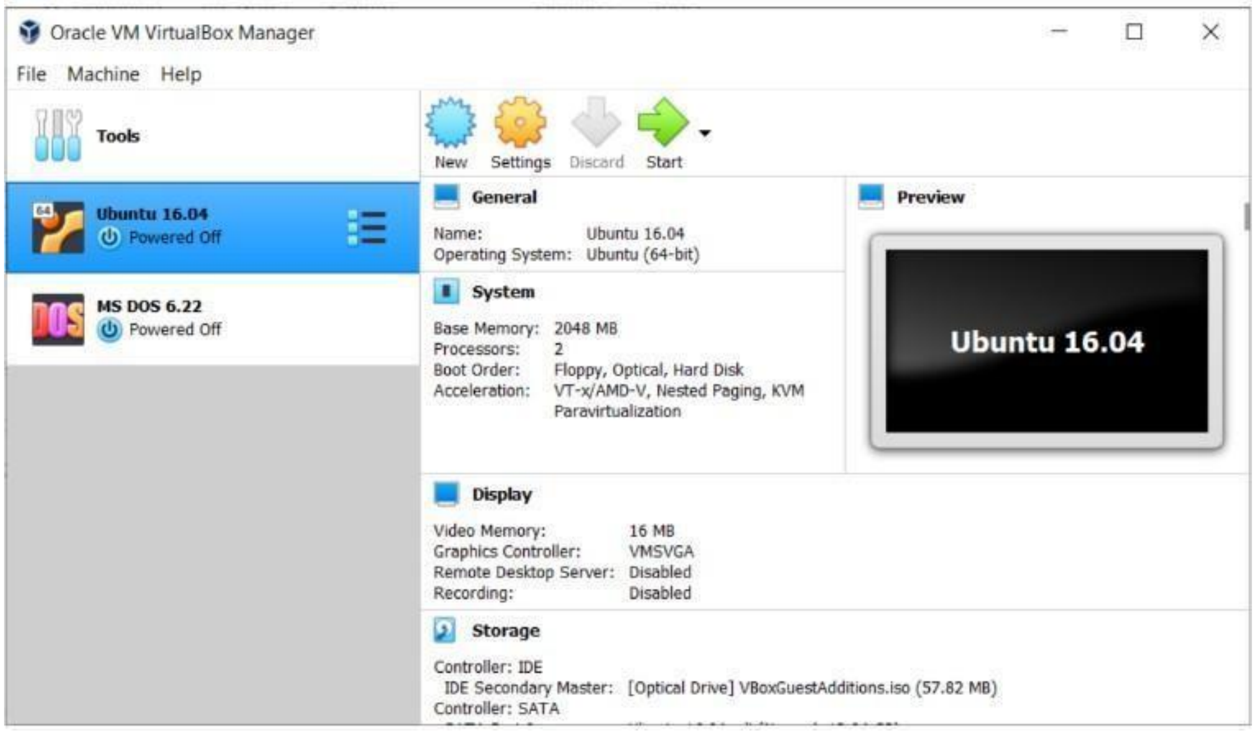
DOS6.22_bootdisk.iso (2.83 MB)

Step 21: Start the machine.



MS DOS 6.22 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
A:\>_
```



General

Name: Ubuntu 16.04
Operating System: Ubuntu (64-bit)

System

Base Memory: 2048 MB
Processors: 2
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Display

Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage

Controller: IDE
IDE Secondary Master: [Optical Drive] VBoxGuestAdditions.iso (57.82 MB)
Controller: SATA

OUTCOME: Different flavors of guest OS (Ubuntu 16.04 and MS DOS) has successfully been installed and launched on the host OS (Windows 10)

EX NO : 2A

DATE:

UNIX COMMANDS

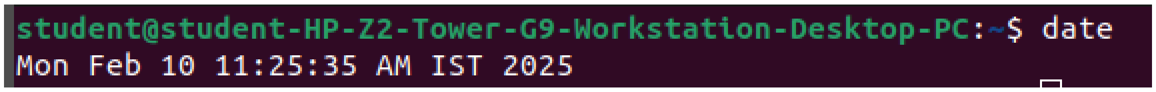
Aim:

To illustrate UNIX commands and Shell Programming

Basic Commands:

DATE

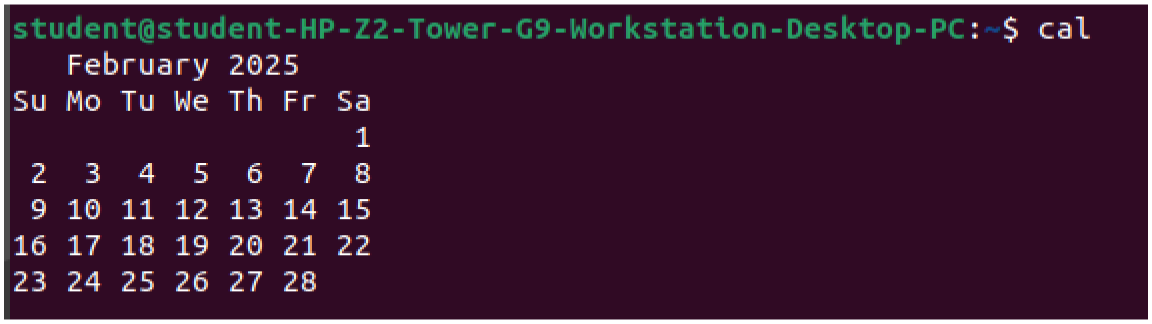
The Unix date command displays or sets the current system date and time, and can format the output in various ways using options and format specifiers.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ date
Mon Feb 10 11:25:35 AM IST 2025
```

CAL

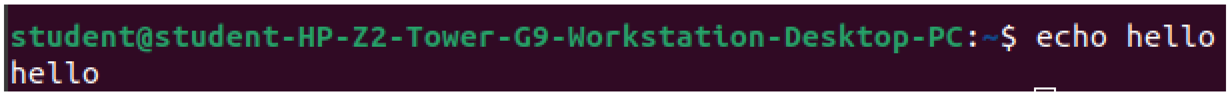
The Unix cal command displays a simple calendar in the terminal, showing the current month by default, with options to view specific months, years, or a full year calendar.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ cal
   February 2025
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
```

ECHO

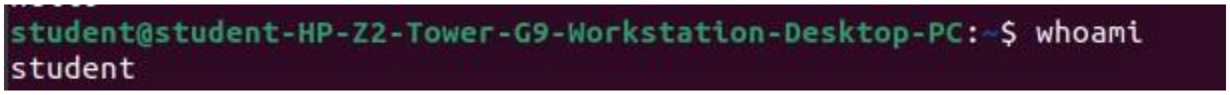
The Unix echo command outputs text or variables to the terminal or a file, commonly used for printing messages or displaying environment variables.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ echo hello
hello
```

WHO AM I

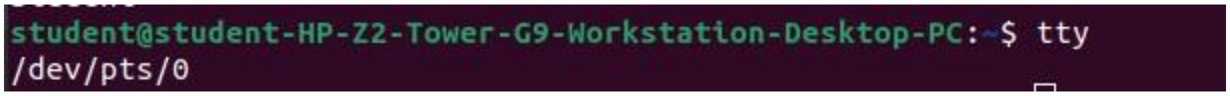
The Unix who am i command displays the username of the current user logged into the terminal session, along with the terminal line, login time, and host information.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ whoami
student
```

TTY

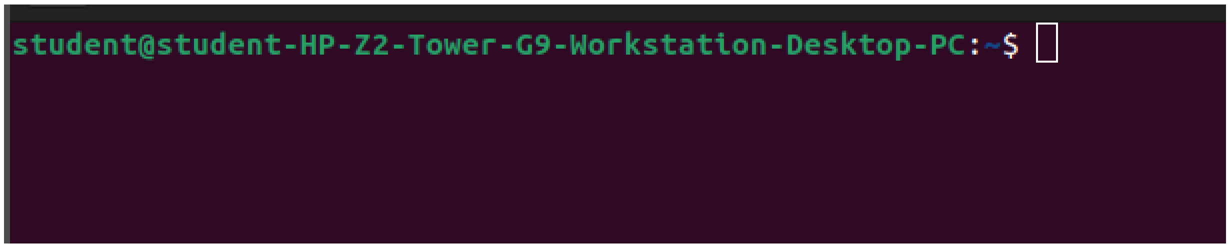
The Unix tty command prints the filename of the terminal connected to standard input, typically showing the device path (e.g., /dev/tty1) for the current terminal session.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ tty
/dev/pts/0
```

CLEAR

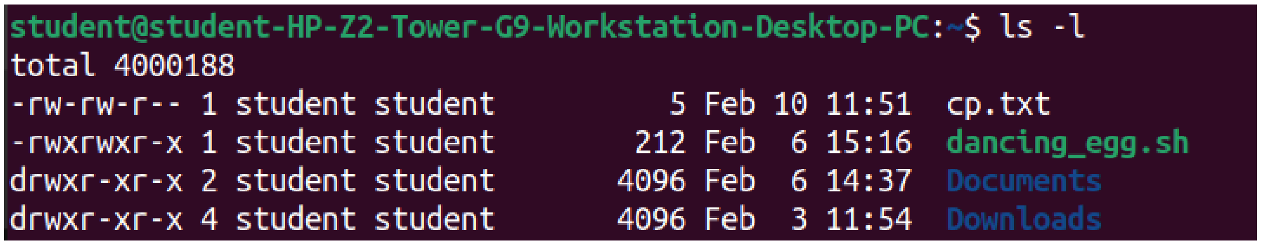
The Unix clear command clears the terminal screen, removing previous output and providing a clean workspace.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$
```

LIST

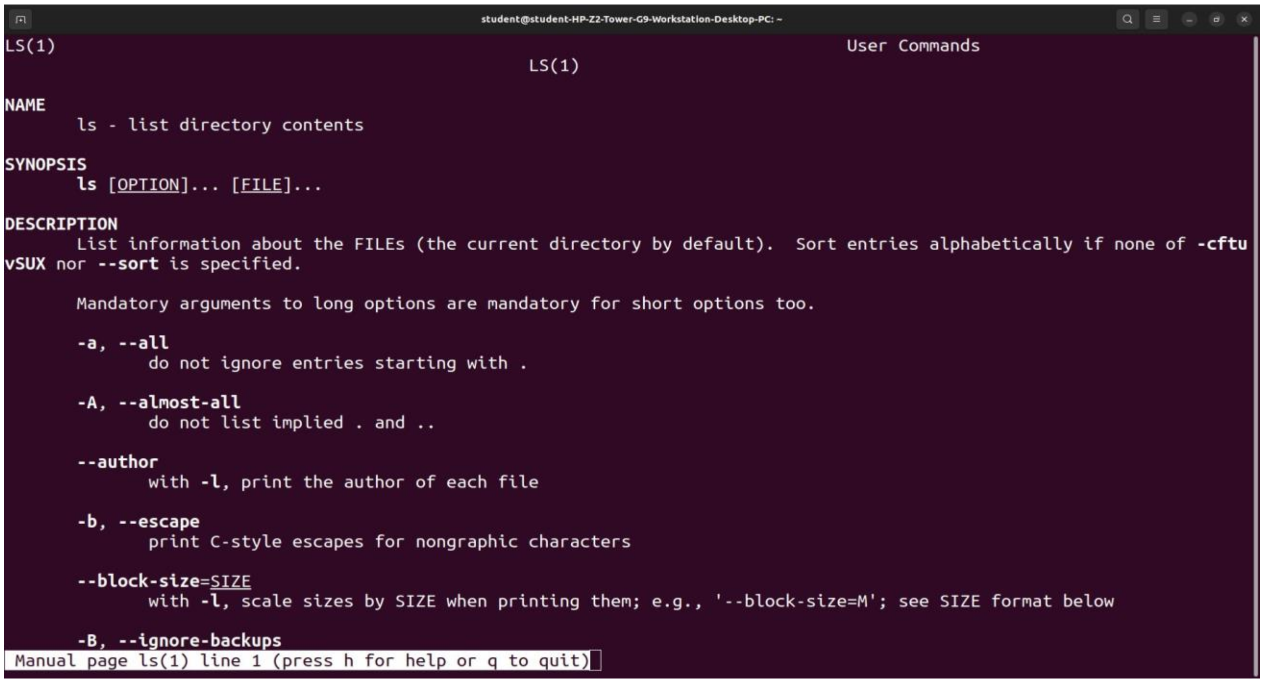
The Unix ls command lists the contents of a directory, such as files and subdirectories, with options to display details like permissions, sizes, and modification dates. For example, ls -l provides a detailed listing.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ ls -l
total 4000188
-rw-rw-r-- 1 student student          5 Feb 10 11:51  cp.txt
-rwxrwxr-x 1 student student        212 Feb  6 15:16  dancing_egg.sh
drwxr-xr-x 2 student student       4096 Feb  6 14:37  Documents
drwxr-xr-x 4 student student       4096 Feb  3 11:54  Downloads
```

MAN

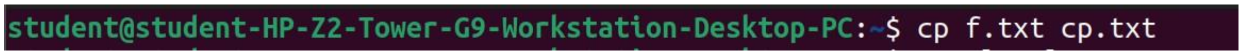
The Unix man command displays the manual pages (documentation) for other commands, providing detailed information about their usage, options, and examples. For example, man ls shows the manual for the ls command.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC: ~
LS(1)                                                                    User Commands
                                                     LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default).  Sort entries alphabetically if none of -cftu
vSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              with -l, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

       -B, --ignore-backups
 Manual page ls(1) line 1 (press h for help or q to quit)
```

COPY

The Unix cp command is used to copy files or directories from one location to another. For example, cp file1.txt file2.txt creates a copy of file1.txt named file2.txt. Use cp -r to copy directories recursively.

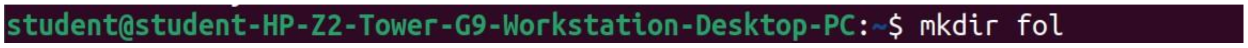


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ cp f.txt cp.txt
```

DIRECTORY RELATED COMMANDS:

Make directory

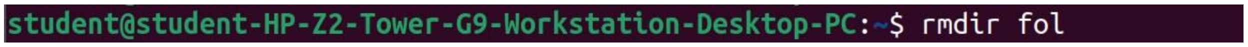
The Unix mkdir command creates a new directory. For example, mkdir new_folder creates a directory named new_folder.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ mkdir fol
```

Remove directory

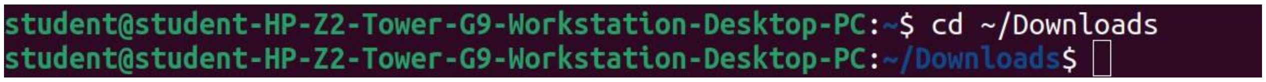
The Unix rmdir command removes an empty directory, while rm -r removes a directory and its contents recursively. For example, rmdir empty_folder or rm -r folder_with_contents.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ rmdir fol
```

Cd

The Unix cd command changes the current working directory. For example, cd /path/to/directory navigates to the specified directory, and cd ~ returns to the home directory.

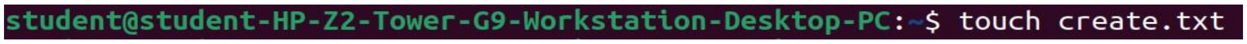


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ cd ~/Downloads
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~/Downloads$ 
```

FILE RELATED:

Create file:

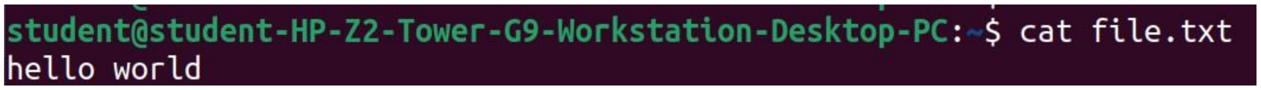
Creates a new, empty file or a file with specified content.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ touch create.txt
```

Display file:

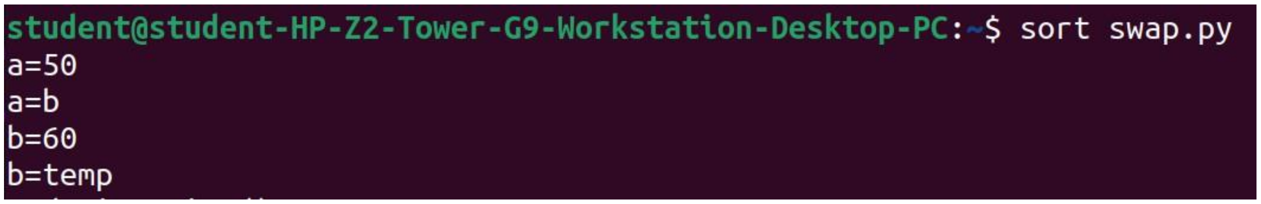
Shows the contents of a file on the screen.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ cat file.txt
hello world
```

Sort:

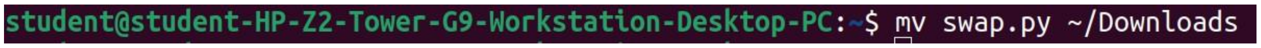
Arranges the lines of a file alphabetically or numerically.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ sort swap.py
a=50
a=b
b=60
b=temp
```

Move a file:

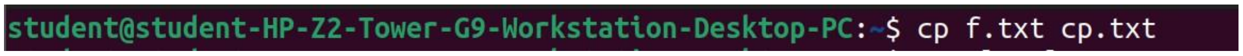
Renames a file or moves it to a different directory.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ mv swap.py ~/Downloads
```

Copy content:

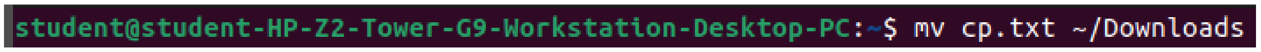
Creates a duplicate of a file's contents in a new file.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ cp f.txt cp.txt
```

MOVE command:

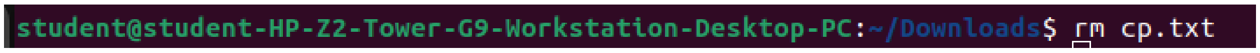
Renames or relocates files and directories.



```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ mv cp.txt ~/Downloads
```

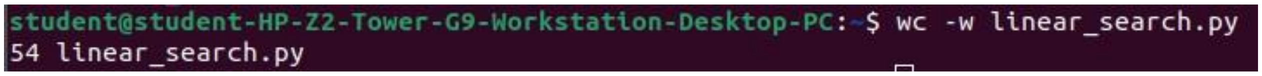
Remove command:

Deletes files or directories (use with caution!).


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~/Downloads$ rm cp.txt
```

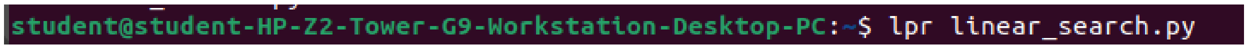
WORD command:

Counts the number of words in a file.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ wc -w linear_search.py
54 linear_search.py
```

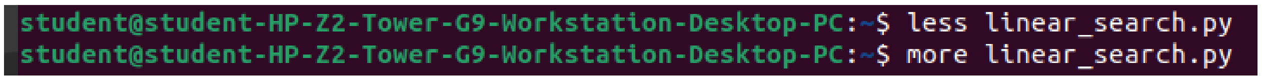
LINE PRINTER:

Sends a file to the printer.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ lpr linear_search.py
```

PAGE command:

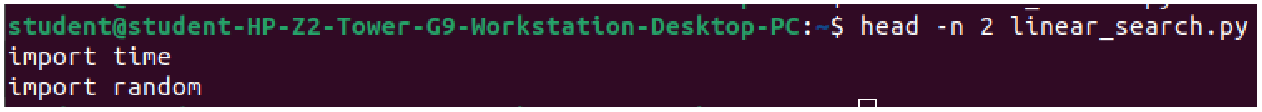
Displays a file's contents one page (or screenful) at a time.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ less linear_search.py
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ more linear_search.py
```

FILTERS AND PIPES

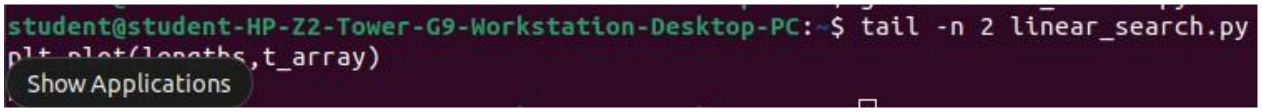
Head

Displays the beginning (first few lines) of the input.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ head -n 2 linear_search.py
import time
import random
```

Tail

Displays the end (last few lines) of the input.

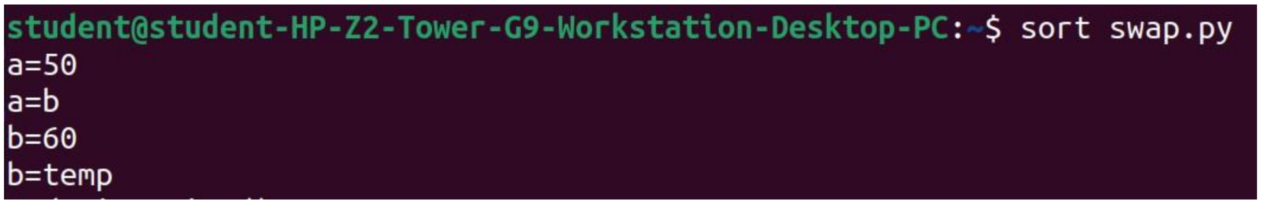

```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ tail -n 2 linear_search.py
plt.plot(lengths,t_array)
```

Grep

Finds and displays lines that match a specific pattern.

Sort

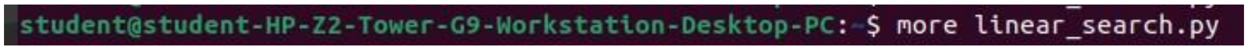
Arranges lines of text in alphabetical or numerical order.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ sort swap.py
a=50
a=b
b=60
b=temp
```

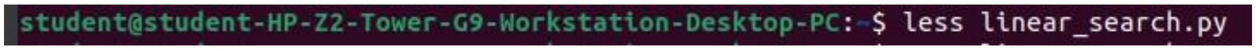
Tr
Replaces or deletes specific characters in the input.

Pipe (|)
Connects the output of one command to the input of another.

More: Displays text one screenful at a time, allowing forward scrolling.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ more linear_search.py
```

Less: Displays text one screenful at a time, allowing both forward and backward scrolling, as well as searching.


```
student@student-HP-Z2-Tower-G9-Workstation-Desktop-PC:~$ less linear_search.py
```

Result:

Unix commands had been runned successfully in the Terminal.

Program:

```
#!/bin/bash

echo "Enter a number: "
read num1
read num2
read num3

if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]; then
        echo "$num1 is greatest."

elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]; then
        echo "$num2 is greatest."

elif [ $num3 -gt $num2 ] && [ $num3 -gt $num1 ]; then
        echo "$num3 is greatest."

else
        echo "All are equal."

fi
```

OUTPUT:

```
Enter a number:
1
1
23
23 is greatest.
```

PROGRAM:

```
#!/bin/bash

n=1
read -p "Enter a number: " num

while [ $num -ge 0 ]; do
    if [ $num -eq 0 ]; then
        n=$((n * 1))
        break
    fi
    n=$((n * num))
    num=$((num - 1))
done

echo "$n"
```

OUTPUT:

```
Enter a number: 8
40320
```

PROGRAM:

```
#!/bin/bash

read -p "Enter a number: " n1
read -p "Enter a number: " n2

a=$((n1 + n2))
b=$((n1 - n2))
c=$((n1 * n2))
d=$((n1 / n2))
e=$((n1 % n2))

echo "ADDITION: $a"
echo "SUBTRACTION: $b"
echo "MULTIPLICATION: $c"
echo "DIVISION: $d"
echo "MODULUS: $e"
```

OUTPUT:

```
Enter a number: 24
Enter a number: 6
ADDITION: 30
SUBTRACTION: 18
MULTIPLICATION: 144
DIVISION: 4
MODULUS: 0
```

PROGRAM:

```
#!/bin/bash

read -p "Enter a number: " num

if [ $((num % 2)) -eq 0 ]; then
        echo "Even"
else
        echo "Odd"

fi
```

OUTPUT:

```
Enter a number: 9
Odd
```

PROGRAM:

```
#!/bin/bash

n="Y"
s=0

while [[ "$n" == "Y" || "$n" == "y" ]]; do
   read -p "Enter a number to add: " num
   s=$((s + num))
   read -p "Do you want to continue [Y/n]? " n
done

echo "Total sum: $s"
```

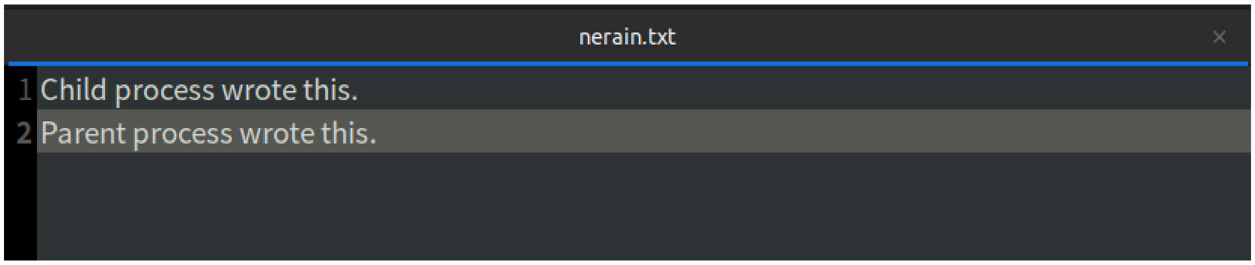
OUTPUT:

```
Enter a number to add: 2
Do you want to continue [Y/n]? y
Enter a number to add: 4
Do you want to continue [Y/n]? y
Enter a number to add: 10
Do you want to continue [Y/n]? y
Enter a number to add: 4
Do you want to continue [Y/n]? n
Total sum: 20
```

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/wait.h>

int main() {
    int fd = open("file.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        perror("Error opening file");
        return 1;
    }

    int pid = fork();

    if (pid == 0) {
        // Child process
        write(fd, "Child process wrote this.\n", 26);
        close(fd);
    } else if (pid > 0) {
        // Parent process
        wait(NULL);  // Wait for child
        write(fd, "Parent process wrote this.\n", 27);
        close(fd);
    } else {
        perror("Fork failed");
        return 1;
    }

    return 0;
}
```

Output:



```
Child process wrote this.
Parent process wrote this.
```

Program:

- **Messaging queue**

Sender code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>

#define MAX_TEXT 512

struct msg_buffer {
    long msg_type;
    char msg_text[MAX_TEXT];
} message;

int main() {
    // Generate unique key
    key_t key = ftok("msgqueue", 65);
    // Create message queue
    int msgid = msgget(key, 0666 | IPC_CREAT);
    message.msg_type = 1;
    printf("Enter message: ");
    // Read message
    fgets(message.msg_text, MAX_TEXT, stdin);
    msgsnd(msgid, &message, sizeof(message), 0);
    // Send message
    printf("Message sent: %s\n", message.msg_text);
    return 0;
}
```

Receiver code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX_TEXT 512
struct msg_buffer {
    long msg_type;
    char msg_text[MAX_TEXT];
} message;
int main() {
    // Same key as producer
    key_t key = ftok("msgqueue", 65);
    // Access message queue
    int msgid = msgget(key, 0666 | IPC_CREAT);
    // Receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);
    printf("Message received: %s\n", message.msg_text);
    // Remove message queue
    msgctl(msgid, IPC_RMID, NULL);
    return 0;}
```

- **Shared Memory**

  Producer code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#define SHM_SIZE 1024  // Shared memory size
int main() {
    key_t key = ftok("shmfile", 65);  // Generate a unique key
    int shmid = shmget(key, SHM_SIZE, 0666 | IPC_CREAT);
    char *data = (char *) shmat(shmid, (void *)0, 0);
    printf("Enter message to write to shared memory: ");
    // Get input from user
    fgets(data, SHM_SIZE, stdin);
    printf("Message written: %s\n", data);
    shmdt(data); // Detach shared memory
    return 0;}
```

  Consumer code:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHM_SIZE 1024  // Shared memory size

int main() {
    key_t key = ftok("shmfile", 65);  // Same key as producer
    int shmid = shmget(key, SHM_SIZE, 0666); // Get shared memory ID
    char *data = (char *) shmat(shmid, (void *)0, 0); // Attach shared memory
    printf("Data read from shared memory: %s\n", data);
    shmdt(data); // Detach shared memory
    shmctl(shmid, IPC_RMID, NULL); // Remove shared memory
    return 0;}
```

OUTPUT:

- **Messaging queue**

```
$ ./sender
Enter message: Using message queue for IPC!!
Message sent: Using message queue for IPC!!
$ ./receiver
Message received: Using message queue for IPC!!
```

- **Shared Memory**

```
$ ./producer
Enter message to write to shared memory: Using shared memory through IPC!
Message written: Using shared memory through IPC!
$ ./consumer
Data read from shared memory: Using shared memory through IPC!
```

PROGRAM:

```
#include <stdio.h>
#define MAX_PROCESSES 10
#define MAX_RESOURCES 10
int n, m; // Number of processes and resources
int alloc[MAX_PROCESSES][MAX_RESOURCES]; // Allocation matrix
int max[MAX_PROCESSES][MAX_RESOURCES];   // Maximum demand matrix
int avail[MAX_RESOURCES];                // Available resources
int need[MAX_PROCESSES][MAX_RESOURCES];  // Need matrix
int safeSequence[MAX_PROCESSES];         // Safe sequence
int finished[MAX_PROCESSES];             // Finished processes tracker
void calculateNeed() {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            need[i][j] = max[i][j] - alloc[i][j];
}
int check(int process) {
    for (int j = 0; j < m; j++)
        if (need[process][j] > avail[j])
            return 0; // Not enough resources
    return 1; // Can proceed
}
int isSafe() {
    int count = 0;
    for (int i = 0; i < n; i++)
        finished[i] = 0; // Mark all processes as unfinished
    while (count < n) {
        int found = 0;
        for (int i = 0; i < n; i++) {
            if (!finished[i] && check(i)) {
                for (int j = 0; j < m; j++)
                    avail[j] += alloc[i][j]; // Release resources
                safeSequence[count++] = i;
                finished[i] = 1; // Mark as finished
                found = 1;}}
        if (!found) // If no process was found in this iteration, deadlock occurs
            return 0;
    }
    return 1;
}
int main() {
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    printf("Enter the number of resources: ");
    scanf("%d", &m);

    printf("Enter the Allocation Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &alloc[i][j]);

    printf("Enter the Maximum Demand Matrix:\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            scanf("%d", &max[i][j]);

    printf("Enter the Available Resources:\n");
    for (int i = 0; i < m; i++)
        scanf("%d", &avail[i]);

    calculateNeed();

    if (isSafe()) {
        printf("The system is in a safe state.\nSafe sequence is: ");
        for (int i = 0; i < n; i++)
            printf("%d ", safeSequence[i]);
        printf("\n");
    } else {
        printf("The system is in an unsafe state! Deadlock may occur.\n");
    }

    return 0;}
```

OUTPUT:

```
Enter the number of processes: 2
Enter the number of resources: 2
Enter the Allocation Matrix:
1 1
1 1
Enter the Maximum Demand Matrix:
9 9
9 9
Enter the Available Resources:
8 9
The system is in a safe state.
Safe sequence is: 0 1
```

Program:

```
#include <stdio.h>
#include <stdbool.h>

int main() {
    int P, R; // Number of processes and resources
    printf("Enter the number of processes: ");
    scanf("%d", &P);
    printf("Enter the number of resource types: ");
    scanf("%d", &R);

    int Available[R];
    printf("\nEnter the Available vector (%d values):\n", R);
    for(int i = 0; i < R; i++) {
        scanf("%d", &Available[i]);
    }

    int Allocation[P][R], Request[P][R];

    printf("\nEnter the Allocation Matrix (%d x %d):\n", P, R);
    for(int i = 0; i < P; i++) {
        for(int j = 0; j < R; j++) {
            scanf("%d", &Allocation[i][j]);
        }
    }

    printf("\nEnter the Request Matrix (%d x %d):\n", P, R);
    for(int i = 0; i < P; i++) {
        for(int j = 0; j < R; j++) {
            scanf("%d", &Request[i][j]);
        }
    }

    // Work = Available
    int Work[R];
    for(int i = 0; i < R; i++) {
        Work[i] = Available[i];
    }

    // Finish array
    bool Finish[P];
    for(int i = 0; i < P; i++) {
```

```
    Finish[i] = false;
}

// Detection algorithm
bool done = false;
while(!done) {
    done = true;  // Assume no process can proceed unless found otherwise

    for(int i = 0; i < P; i++) {
        // Check if this process can still proceed
        if(!Finish[i]) {
            bool canProceed = true;
            for(int j = 0; j < R; j++) {
                if(Request[i][j] > Work[j]) {
                    canProceed = false;
                    break;
                }
            }

            if(canProceed) {
                // This process can proceed, so mark it as finished
                Finish[i] = true;
                done = false; // We found a process that can proceed; keep going

                // Release its resources to Work
                for(int j = 0; j < R; j++) {
                    Work[j] += Allocation[i][j];
                }
            }
        }
    }
}

// Check for any unfinished processes
bool deadlock = false;
printf("\nProcesses in Deadlock (if any):\n");
for(int i = 0; i < P; i++) {
    if(!Finish[i]) {
        printf("Process %d is in deadlock.\n", i);
        deadlock = true;
    }
}

if(!deadlock) {
        printf("No deadlock detected.\n");
    }

    return 0;
}
```

Output:

```
Enter the number of processes: 3
Enter the number of resource types: 3

Enter the Available vector (3 values):
2 1 0

Enter the Allocation Matrix (3 x 3):
0 1 0
2 0 0
1 0 1

Enter the Request Matrix (3 x 3):
2 0 0
0 1 0
0 0 0

Processes in Deadlock (if any):
No deadlock detected.
```

PROGRAM:

```
#include <stdio.h>
void firstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n]; // Stores block index allocated to each process

    for (int i = 0; i < n; i++)
        allocation[i] = -1; // Initially, no process is allocated

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                allocation[i] = j;
                blockSize[j] -= processSize[i]; // Reduce available block size
                break;
            }
        }
    }
    printf("\nProcess No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf(" %d\t\t%d\t\t", i+1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");}}
int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    printf(“First Fit Allocation:\n”);
    firstFit(blockSize, m, processSize, n);
    return 0;}
```

OUTPUT:

```
First Fit Allocation:
Process No.    Process Size   Block No.
 1             212            2
 2             417            5
 3             112            2
 4             426            Not Allocated
```

PROGRAM:

```
#include <stdio.h>
void bestFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;
    for (int i = 0; i < n; i++) {
        int bestIndex = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (bestIndex == -1 || blockSize[j] < blockSize[bestIndex])
                    bestIndex = j;}}
        if (bestIndex != -1) {
            allocation[i] = bestIndex;
            blockSize[bestIndex] -= processSize[i];}}
    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");}}
int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    printf("Best Fit Allocation:\n");
    bestFit(blockSize, m, processSize, n);
    return 0;}
```

OUTPUT:

```
Best Fit Allocation:
Process No.     Process Size    Block No.
1               212             4
2               417             2
3               112             3
4               426             5
```

PROGRAM:

```
#include <stdio.h>
void worstFit(int blockSize[], int m, int processSize[], int n) {
    int allocation[n];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;
    for (int i = 0; i < n; i++) {
        int worstIndex = -1;
        for (int j = 0; j < m; j++) {
            if (blockSize[j] >= processSize[i]) {
                if (worstIndex == -1 || blockSize[j] > blockSize[worstIndex])
                    worstIndex = j;}}
        if (worstIndex != -1) {
            allocation[i] = worstIndex;
            blockSize[worstIndex] -= processSize[i];}}
    printf("Process No.\tProcess Size\tBlock No.\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t", i + 1, processSize[i]);
        if (allocation[i] != -1)
            printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");}}
int main() {
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);
    printf("Worst Fit Allocation:\n");
    worstFit(blockSize, m, processSize, n);
    return 0;}
```

OUTPUT:

```
Worst Fit Allocation:
Process No.     Process Size    Block No.
1               212             5
2               417             2
3               112             5
4               426             Not Allocated
```

PROGRAM:

```
#include <stdio.h>

int main() {
    int n, timeQuantum;
    printf("Enter number of processes: ");
    scanf("%d", &n);

    int burstTime[n], remainingTime[n], waitingTime[n], turnaroundTime[n];
    int i;
    for (i = 0; i < n; i++) {
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &burstTime[i]);
        remainingTime[i] = burstTime[i];
        waitingTime[i] = 0;  // initialize waiting time to 0
    }

    printf("Enter time quantum: ");
    scanf("%d", &timeQuantum);

    int time = 0;  // current time
    int done;

    do {
        done = 1;
        for (i = 0; i < n; i++) {
            if (remainingTime[i] > 0) {
                done = 0;
                if (remainingTime[i] > timeQuantum) {
                    time += timeQuantum;
                    remainingTime[i] -= timeQuantum;
                } else {
                    time += remainingTime[i];
                    waitingTime[i] = time - burstTime[i];
                    remainingTime[i] = 0;
                }
            }
        }
    } while (!done);

    for (i = 0; i < n; i++) {
        turnaroundTime[i] = burstTime[i] + waitingTime[i];
    }
    printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", i + 1, burstTime[i], waitingTime[i], turnaroundTime[i]);
    }

    return 0;
}
```

OUTPUT:

```
Enter number of processes: 7
Enter burst time for process 1: 45
Enter burst time for process 2: 23
Enter burst time for process 3: 78
Enter burst time for process 4: 45
Enter burst time for process 5: 89
Enter burst time for process 6: 90
Enter burst time for process 7: 12
Enter time quantum: 4

Process     Burst Time     Waiting Time     Turnaround Time
1           45             211              256
2           23             116              139
3           78             277              355
4           45             216              261
5           89             291              380
6           90             292              382
7           12             72               84
```